Common Logistics Command and Control System

CLC2S 2.0 Application Architecture Overview

Office of Naval Research

Naval Facilities Engineering Service Center

Marine Corps Systems Command

Sapient Corporation

20 April 2004

Table of Contents

Application Architecture

Introduction

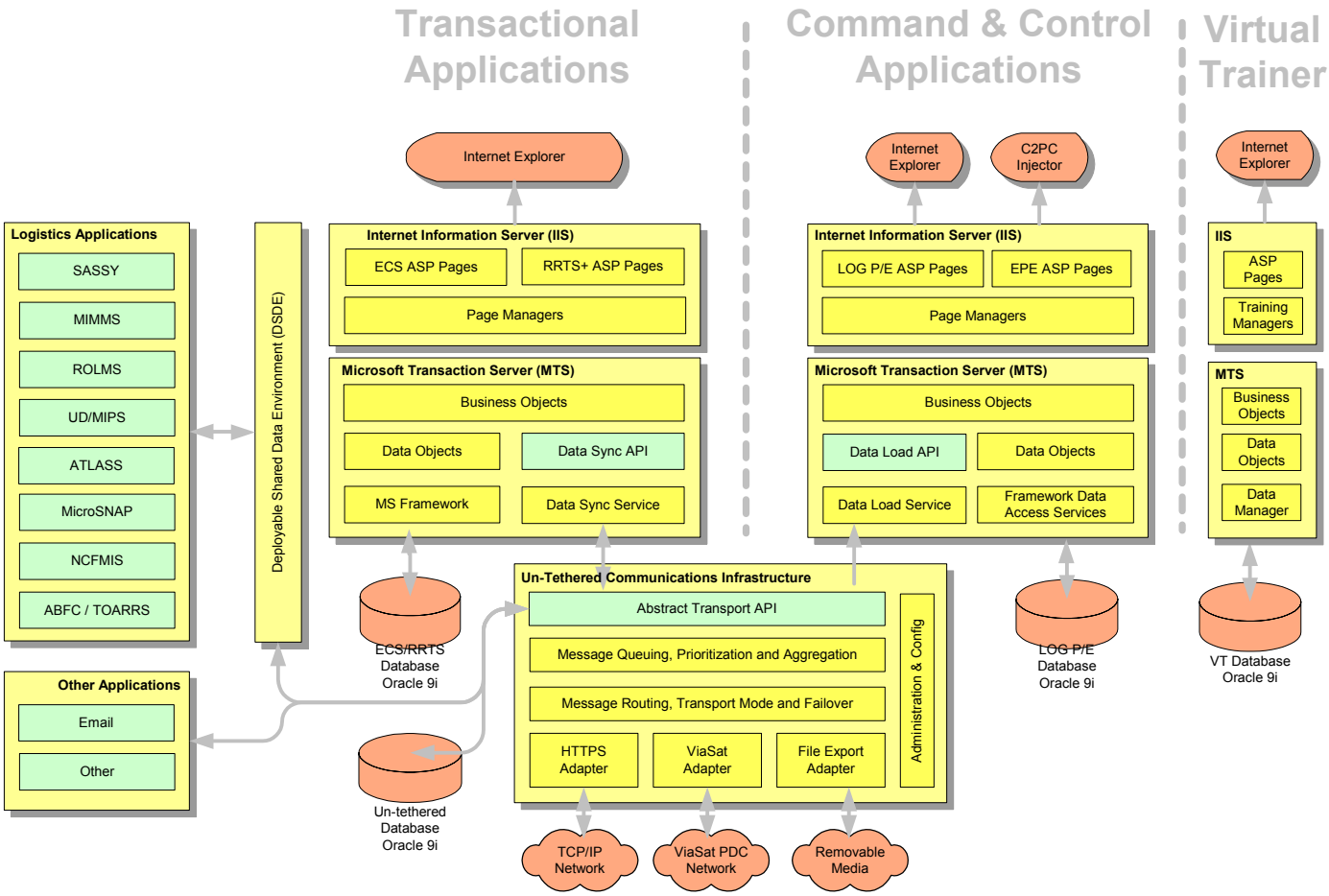
This section describes the application architecture for the CLC2S 2.0 application. It covers a variety of areas, ranging from a high-level architecture description to justifications for specific technical choices.

This document begins with a description of the high level architecture. Next, the technology approach section defines both the tools selected and some details of their use. Next, network security issues are addressed. Finally, topics whose resolution will provide the correct direction for further developments are discussed in the “Areas to Explore” section.

High Level Architecture

The CLC2S system uses traditional 3-tier web architecture. As shown below, most communications with the system occurs through the web or application server, either directly or via. the Deployable SDE and/or Un-Tethered Communications Infrastructure.

CLC2S 2.0 Application Architecture



The CLC2S system is comprised of five modules, some of which in fact built on the same application stack:

- Enhanced CSSOC/COC System (ECS) – ECS/RRTS application stack
- Rapid Request Tracking System (RRTS+) – ECS/RRTS application stack
- Logistics Planning and Execution (Log P/E) – LogP/E application stack
- Engineering Planning and Execution (EPE) – LogP/E application stack
- Virtual Trainer – VT application stack.

Regardless of the specific stack, all of the modules share the same web server, which generates the pages and sends them to using units. Each has its own set of business components that provides logistics functionality, and each has its own Oracle 8/9i database instance in which their respective tables are stored. A system of DSDE XML based interfaces and Un-Tethered Communication APIs exist which enable data to be imported from ECS/RRTS into LogP/E and EPE as well as to/from other applications. The integration between the ECS/RRTS+ and Log P/E stacks ensures that changes in the transactional ECS/RRTS systems are reflected in the LogP/E and EPE DSS system.

Whilst there are 4 logical servers, (web server, COM+ application server, two database servers) all can reside on one physical server.

The server is connected to the local area network (LAN) within the CSSE. Using units within the CSSE connect to the web server directly through the LAN. Using units in other areas, such as the MEF, ACE, and GCE, with digital communication connections to the LAN (via MUX, HF, satellite, or other technology) also connect in a similar manner. However, these remote using units have less bandwidth to the server and response times may suffer. Application design accounts for these bandwidth limitations and steps are taken to minimize its effects.

In general, data from legacy systems (ODSE, ROLMS, SASSY, MDSS II, etc.) is imported into the ECS/RRTS+ database from flat file dumps from other systems via. the XML interfaces provided by the DSDE. These communications are typically through the CSSE LAN, though that is not a requirement.

Technology Approach

To both provide application scalability and minimize administrative needs, the CLC2S system uses Microsoft web technology and an Oracle database. This choice allows the entire system to run on a single physical server, though still permits the use of multiple machines for greater performance and larger numbers of users. All web and application server software is standard with Windows NT and 2000 Servers. Only WinNT/2000 and Oracle administration skills are needed to install and run the system.

The Microsoft technologies employed by the CLC2S system are Internet Information Server (IIS) and Microsoft Transaction Server (MTS/COM+). Microsoft Visual Basic (VB) is used to create Component Object Model (COM) objects which are employed as either business or display objects and managed by MTS/COM+. MTS/COM+ provides such functionality as database connection pooling, load balancing, transaction management, and logging.

ECS/RRTS was originally developed in ColdFusion prior to being ported to the baseline CLC2S platform. The question of why MTS/COM+ was used instead of ColdFusion is often raised. ColdFusion is an application server that is tightly tied to HTML. Because it is so tightly coupled to HTML, there is no separation of business logic from display logic.

Because of this, ColdFusion applications can only be deployed to client machines that have a full-fledged web browser. Many potential interfaces to the system (eg, in vehicle systems etc.) do not have this capability. By separating the business logic from the display logic, custom display logic can be written for different types of devices, while all of those devices access the same business logic, allowing a much wider range of devices to deploy to.

Continued deployment with ColdFusion would also have required a license for each ECS/RRTS server and additional training for administrators resulting in increased cost for each deployment.

Performance of the system is determined by many factors including number of processors and servers, structures of disk arrays, amount of memory within the servers, and amount of data within the database. Please refer to SV7 - System Performance Parameters document for more details.

The CLC2S system was loaded with the data from DK/SK01 and NCF FEX Bearing Duel to give a representative level of data within the system and then tuned for performance on the worst case scenario configuration of all parts of the system being installed on a single workstation with a single processor. The system, during testing has not encountered any problem with locking, transaction handling and module interaction despite intensive load testing with a variety of testing tools (see SV8 for details). The entire application was designed for reasonable throughput and certain steps were followed:

- The graphics have been minimized in the screen layouts, so as to avoid a heavy “download” time, keeping in mind the limited network bandwidth.
- Functions have been added to the application code to log timestamps for execution and have been logged to files on the server – accessible via the CLC2S Administrative Console GUI.
- The execution plan of the database queries has been investigated and tuned for optimum performance. The data was made as realistic as possible and indexes have been added, (wherever needed) to speed up the execution.
- Access to the Oracle database is via. stored procedures wherever possible to improve database performance.

On the server, the majority of resources are utilized by Oracle. Queries are run in the order received on the resources available. On a single processor machine, this means that a query cannot be started until the database has finished processing any prior queries. However, if the server is a multiple processor machine, multiple queries can be executed at the same time. If a user executes a query that takes a long time, queries from other users will pool up and the users will have to wait in order of the other users to get their results. Because of this, the minimum hardware recommendation for the machine used to run the Oracle database is at least a Pentium III with 256MB RAM.

Communications outside of the CSSE LAN will be much slower than communication directly over the CSSE LAN. The following is a comparison of the time required to download 15K over various dedicated bandwidths:

Bandwidth	Standard Name	Download Time (s)
10Mbps	Thin Ethernet, category 3 cable, cable modem	0.0123
6.144 Mbps	Standard ADSL downstream	0.02
1.544 Mbps	**T-1, DS-1 North America**	0.0796
128 Kbps	ISDN	0.96
56 Kbps	56flex, U.S. Robotics x2 modems,	2.194
	56flex, x2 modem communications rate	3.657
28.8 Kbps	V.34, Rockwell V.Fast Class modems	4.267
14.4 Kbps	V.32bis modem, V.17 fax	8.533
9600 bps	modem speed circa 1980s	12.8
2400 bps	modem speed circa 1980s	51.2

Users operating outside of the CSSE LAN will not affect users within the CSSE LAN any more than if they were within the CSSE LAN themselves.

Network Security Concerns

The United States Marine Corps and United States Navy maintain a strict doctrine regarding requirements surrounding network security. Options for the architecture of the system to be built during the SUL ACTD (Advanced Concept Technology Demonstration) were partially driven by these requirements in order to work with existing systems. On September 14th, 1999 a meeting was held at Quantico to discuss those requirements. The following summarizes the services allowed over the network and how compliance with these requirements impacts system deployment.

Services Relevant to CLC2S

- HTTP – inbound traffic by proxy with strong authentication currently allowed, but not in the future; outbound traffic by proxy and IP filtering. Publicly accessible information should be placed on a server outside the firewall in the future.
- SHTTP – permitted via the HTTP proxy
- FTP – inbound traffic by strong authentication currently, not allowed in the future; outbound by proxy
- SQL*Net – currently, inbound traffic allowed by proxy with strong authentication, outbound by stateful packet filter proxy; in the future not allowed
- NETBIOS – not allowed now or in the future
- RPC – not allowed now or in the future
- NFS – not allowed now or in the future
- JavaScript – allowed for non-critical cosmetic functionality only
- Java – not allowed now or in the future
- ActiveX – not allowed now or in the future
- SMQP – not allowed now or in the future

Services to and from a static IP address can be allowed through the firewall, however such firewall policy decisions are discouraged. Base to base communication is performed over a VPN and PKI is deployed at base sites.

General Impacts

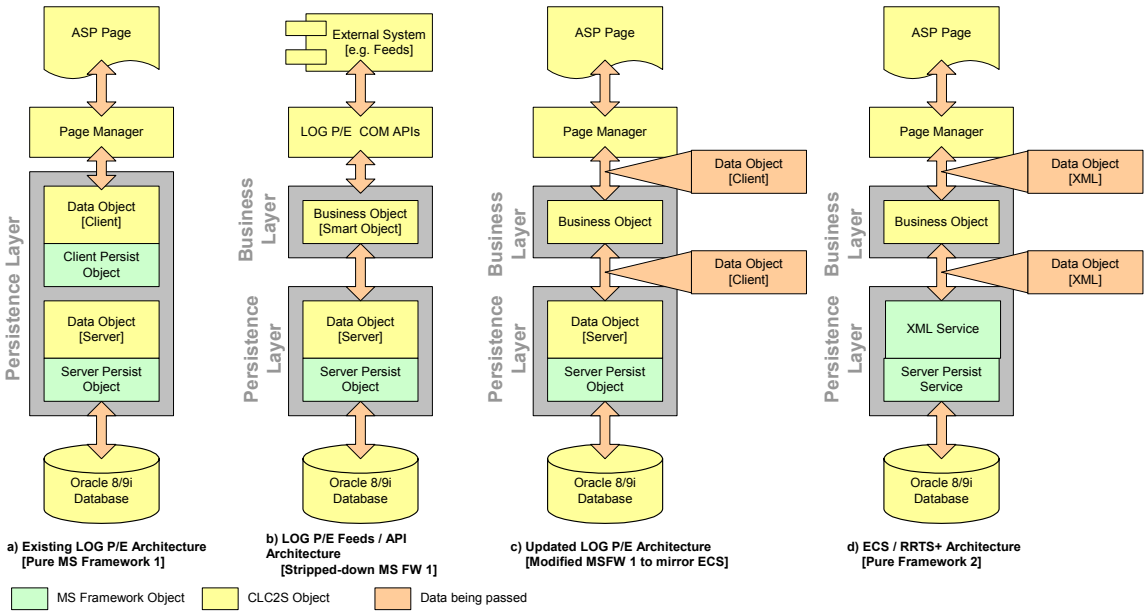
Due to the fact that in the future both inbound HTTP traffic and all SQL*Net traffic will not be allowed, the system will not be able to be accessed from beyond the firewall unless changes are made to the firewall. If the system is only to be used within and between bases, communication between systems should not be an issue due to the VPN implementation and PKI deployment.

By requiring the CLC2S system to only transmit HTML, most firewall issues are resolved. If a single server is to support more than one base, then there are two possibilities. First, the server can be hosted outside of USMC firewalls. This is an immediately viable solution. Second, a VPN solution can be installed at the bases that share a server so the inbound HTTP requests will not be an issue. The USMC is currently investigating the use of VPN for base-to-base communications, though an implementation schedule has not been defined.

Logical Architecture

The logical application architectures of the CLC2S system is as follows:

1) CLC2S Application Architectures

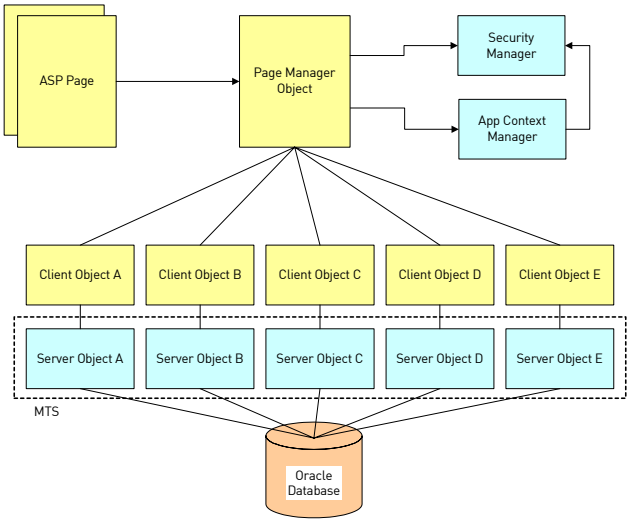


The design of data objects follows Microsoft Framework design. The server object is a transactional object that runs under the MTS runtime environment. The client object is a stateful, non-transactional COM object. The threading model used for each object is the Apartment Threading (MS) model that means that each thread has its own copy of global data and hence eliminates any threading conflicts. However it also means that data cannot be shared between objects. Please refer to the Microsoft framework (versions 1 and 2) documentation for further details.

Page Managers

Each ASP page instantiated its page manager object upon being parsed by the IIS. The PM object is then initialized with the ASP intrinsic objects (namely session, request, response, server, etc.).

Each HTML page is divided into 2 parts: static Html and dynamic Html. The static Html code contains the skeleton of the page (e.g. Table layout, formatting style, etc.) while the dynamic Html code contains the data that are being queried from the DB. The static Html code is normally placed in the ASP file and the page manager object generates the dynamic Html code. This design provides the flexibility when we wish to change the page layout:



Security Design

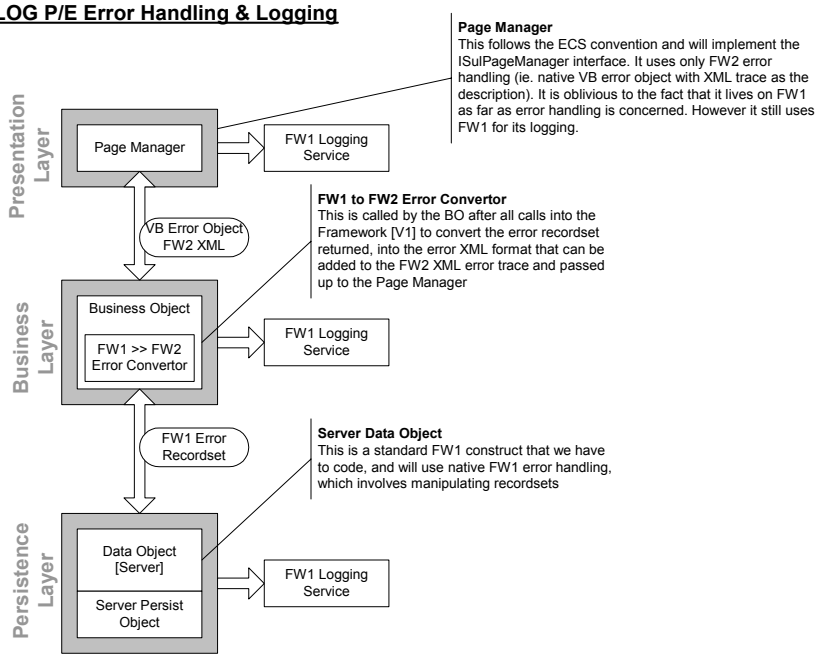
Uses the security manager object and context manager to validate if the user is logged on. Use of context ensures that the user reaches a particular page in proper sequence. Each of the requirements determination pages sets the context and information about which other pages can browse to this page is stored in database.

Error and Exception Handling Strategy

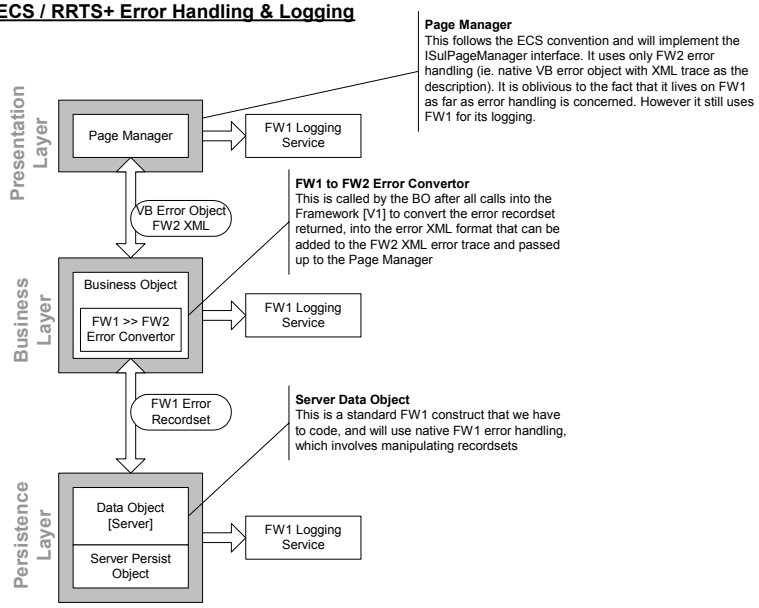
Each method within the business and data objects has either a ADO recordset or XML document (depending on the framework version), which is used to pass back the details of an error if it occurs. The business objects and page managers can use this recordset to extract the description along with the location of where the error occurs.

See the diagrams below for specifics on how MS Framework 1 and 2 perform the exception handling:

2) LOG P/E Error Handling & Logging



3) ECS / RRTS+ Error Handling & Logging

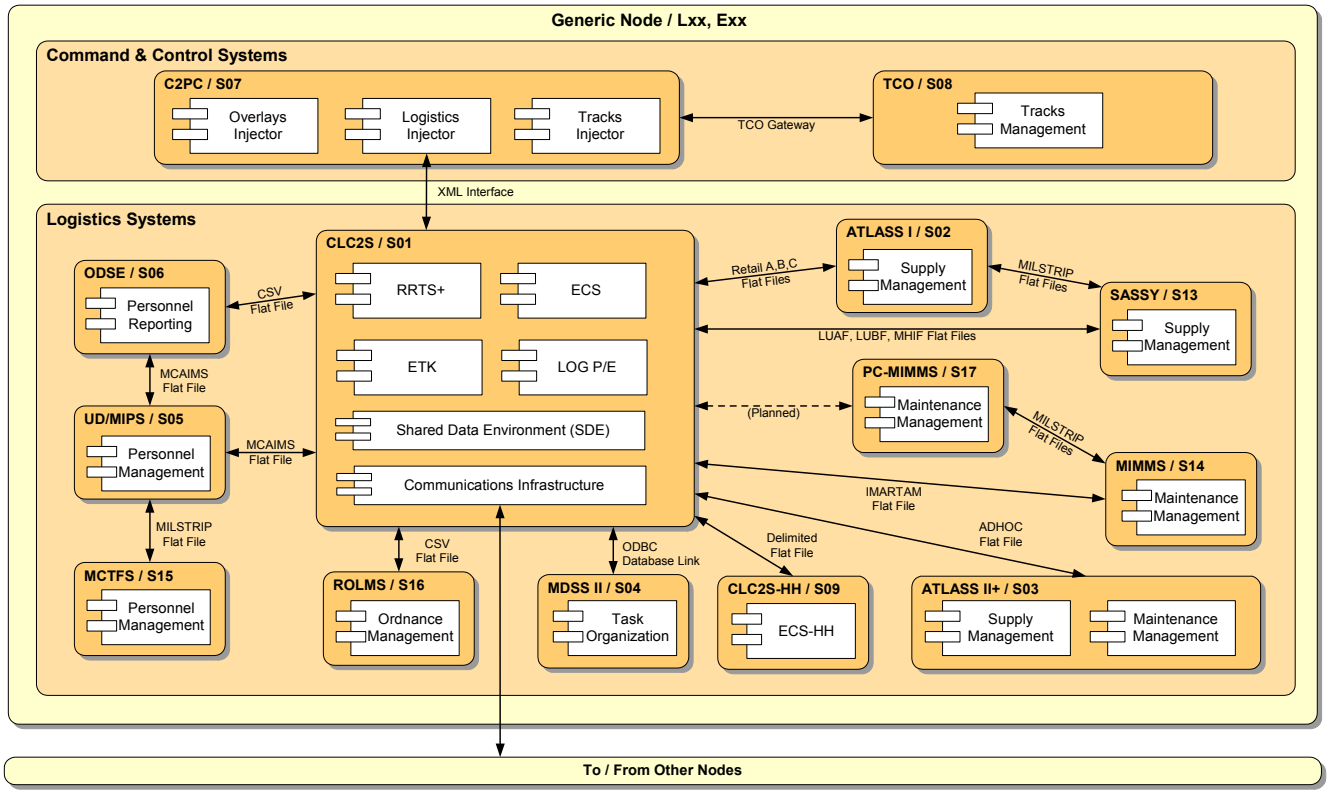


Integration With CLC2S

The concept of “integration” with CLC2S can be looked at in a number of ways, depending on the scope at which you look at the system. At the highest level, CLC2S is a totally standalone system that passes data with other Navy and Marine Corps systems; at a lower level, modules of CLC2S integrate with each other, such as ECS feeding data into Log P/E. The approach to integrating with CLC2S first depends on the scope of the desired integration.

This section describes the various integration “touch points” by looking at CLC2S at varying levels of abstraction.

High-Level Interface View



At the highest level, all of CLC2S can be treated as a single “node” that needs to integrate with other Navy or USMC systems. The following diagram is taken from the CLC2S C4ISR documentation; specifically the “System View 1 -

At this level, the integration with CLC2S usually consists of passing data back and forth with other systems rather than any kind of programmatic integration that is based on code or function calls. As the diagram indicates, this primarily involves passing around data files in the form of individual files as either flat text or XML.

System interfaces available at this level include:

- Deployed Shared Data Environment (DSDE)
- C2PC Logistics Injector
- Reporting Engine

Deployed Shared Data Environment (DSDE)

The Deployed Shared Data Environment (DSDE, also sometimes referred to as the Lightweight SDE or LSDE) was introduced in CLC2S version 2.0 and provides a robust solution for getting data into and out of CLC2S. This can range from one-time data loading done to stand up an instance of CLC2S all the way to an integrated sharing of data between CLC2S and an external system.

Whenever possible, any integration that requires sharing of data between CLC2S and another system should make use of the DSDE. The DSDE administrative tools and functions are designed to provide a great deal of flexibility and visibility into these processes, and therefore require less administrative support and maintenance.

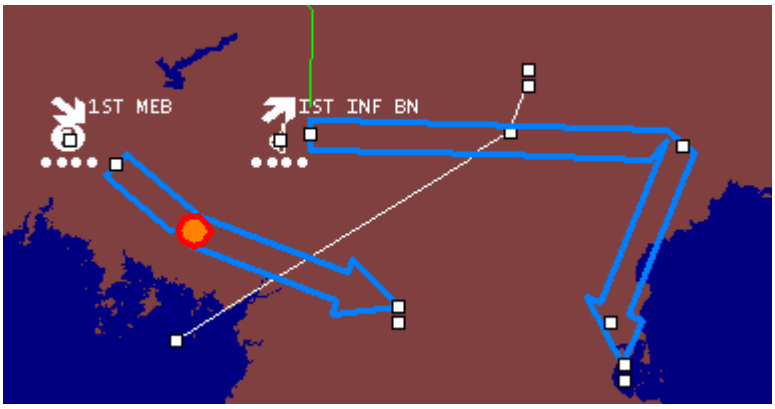
The primary limitation of the DSDE is that it deals with data loading in a largely asynchronous manner. Unlike a web service (for example), the DSDE does not provide programmatic hooks for another system to call for processing data; instead, data is passed back and forth in the form of flat text files. To allow for tighter integration, the CLC2S admin tool can set schedules for these text files will be generated or processed (e.g. generate a file containing personnel updates every 10 minutes). This approach might be used to approximate a synchronous transaction, but a true synchronous interface to the system would require coding at a much lower level.

References and Resources

- **Deployed_SDE_User_Guide.doc** – Detailed reference and how-to guide for using the DSDE, which includes setting up, running and maintaining feeds to/from various systems.
- **AdminApp_1.0 Overview.pdf** – PowerPoint guide describing the usage of the CLC2S administrative tool, which provides access to the DSDE functions.

C2PC Logistics Injector

The Logistics Injector is a plug-in to C2PC that creates a map overlay for displaying CLC2S data in a graphical format. While a standard C2PC map may show terrain data and unit movements, the Logistics Injector also overlays unit readiness data on the units and allows a user to create Log P/E CSS support mission plans graphically on the map.



From an integration perspective, the injector is obviously designed with C2PC in mind, but the mechanism for moving the data between CLC2S and C2PC could also be extended for other purposes. Because the C2PC client and CLC2S installation could be installed on different machines, the injector provides a mechanism in some ways similar to a web service that retrieves mission and unit readiness data and passes it back and forth as XML files.

Unlike the DSDE (which plugs into ECS and RRTS+), the Logistics Injector retrieves and modifies data within Log P/E. Since it is limited to unit readiness and mission planning data, the XML schemas are inherently more limited, but (also unlike the DSDE) it does so in a more synchronous fashion. The data is passed back and forth via XML, but there also exists a programmatic interface for retrieving and storing the data.

References and Resources

- **CLC2S Logistics Injector for C2PC Installation Guide.pdf** – Describes the installation and configuration of the Logistics Injector.
- **C2PCLogisticsInjectorUserGuide.doc** – Detailed user guide for all of the functionality of the Logistics Injector. This also describes at a high level the XML schema used for passing unit readiness and mission planner data back and forth.
- **C2PCLogisticsInjectorDesignDocument.doc** – Detailed technical design document explaining the Logistics Injector at a code level.

Reporting Engine

The CLC2S reporting engine represents another mechanism for getting data out of the system. Designed to make use of Microsoft Access, reporting data is therefore highly portable (in the form of an Access MDB file), and allows users to use the Access reporting tools to quickly create their own reusable custom reports.

Internally, CLC2S uses an Oracle database that contains all the data relevant to the system and is usually (though not always) resident on the CLC2S machine. This
database includes all of the data used by the application and therefore includes all the personnel, equipment, supplies and unit data displayed by the system. However, the Oracle installation also takes up several gigabytes of space and contains system configuration data and low-level program logic which, if modified, can potentially have catastrophic effects on the system.

The reporting engine moves data from the Oracle database directly into an Access database (referred to as the "Data Mart"), and contains only the dynamic user-facing data that would be of interest to an end user (i.e., personnel, equipment, supplies, units, etc.). It also converts the internal Oracle structure to one that is more user-friendly, denormalizing the data and synthesizing data from several database tables into a single, coherent view. (For instance, Supply data in Oracle is spread out over several tables, linked together by key values that may not make sense to an end user. The reporting database consolidates these tables into a single table with user-friendly column names.)

From an integration perspective, the reporting engine obviously has a number of limitations. First, it is only one-way: data can only come out of CLC2S. Also, it's not as configurable as the DSDE, since it will export all the relevant CLC2S data rather than just a subset of "interesting" data (personnel data, for instance). For these reasons, it is generally better to use the DSDE functionality whenever possible. However, for situations where a wide breadth of CLC2S data needs to be exported in a portable format - for instance for reporting trends over time - this may present a viable alternative.

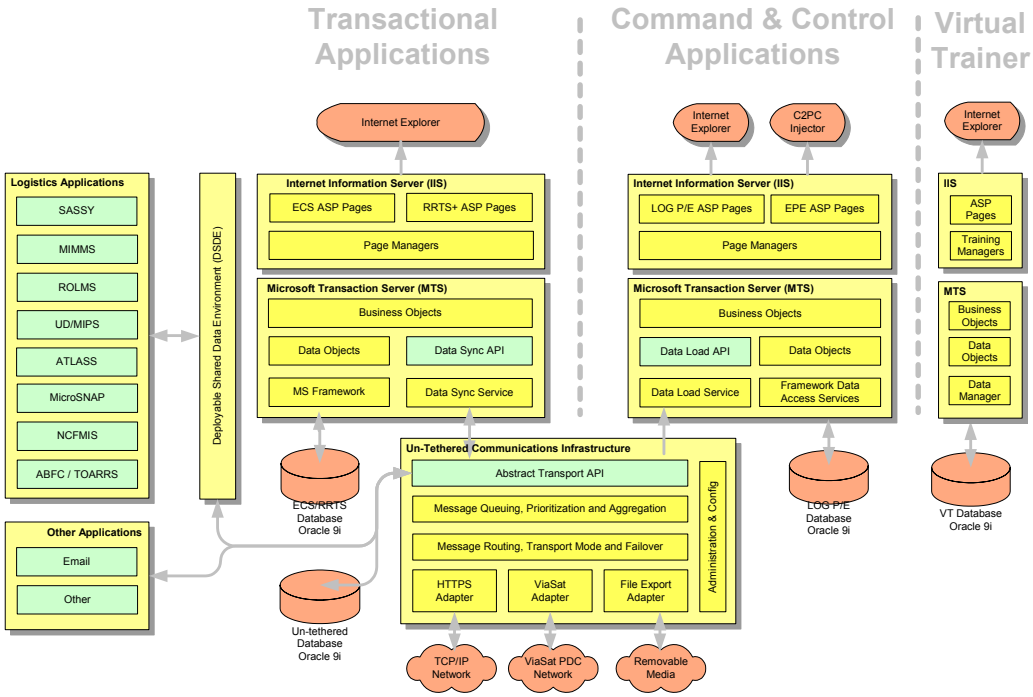
References and Resources

- **ReportingTool_1.Overview.ppt** – Mid-level overview of the reporting tool, designed for teaching and a reference guide.
- **Reports_User_Guide.doc** – Detailed description of the reporting tool; this describes the process for how the reporting engine works and the reporting schema that is used to retrieve the data from Oracle.

Low-Level Interface View

Viewing CLC2S at a lower level, it is possible to view the overall system as discrete modules, such as between the transactional applications (ECS and RRTS+) and the C2 applications (Log P/E and Eng P/E). At this level, it is possible to see how these modules interact with each other, which in turn exposes more integration points into the system.

CLC2S 2.0 Application Architecture



From this view, it is apparent that at a low level CLC2S is broken into three main areas:

- Transactional Applications (ECS and RRTS+)
- Command and Control Applications (Log P/E and Eng P/E)
- Communications Infrastructure (Un-tethered and DSDE functionality)

From an external user's perspective, these parts operate together seamlessly. For instance, data changes in ECS are reflected in the unit readiness meatballs in Log P/E, and may be communicated to other un-tethered nodes or external systems without the need for user intervention. However, this breakdown is instructive when considering integration points into CLC2S. For instance, it may be desirable to only retrieve mission planner data; in this instance, it would be more effective to focus integrating only with the C2 applications rather than the transactional applications.

Unlike the higher-level integration points described earlier, system interfaces at this level usually require some degree of programmatic interface. That is, instead of simply passing data back and forth, it is also necessary to utilize an Application Programming Interface (API) through code that sends the data back and forth. This approach is much more complex but can also provide lower-level hooks into the system. This may be useful (or even required) when adding new functionality to the system, for example.

System interfaces available at this level include:

- Log P/E API
- Bill Of Materials (BOM) Interface for Eng P/E
- Un-tethered API
- Direct Business Object access
- Direct database access

Log P/E Application Programming Interface (API)

The separation between the transactional applications (ECS and RRTS+) and the C2 applications (Log P/E and Eng P/E) is more than just logical. Although both exist on the same machine and use the same Oracle database, the data and program logic is segregated to keep their functionality separate and distinct.

One side effect of this is that since data must still be kept in synch (for instance, ECS data drives the unit readiness data displayed in Log P/E), a process exists in CLC2S to automatically move the data from one stack to the other. On the ECS/RRTS+ side, this utilizes the DSDE export process to get the relevant updated data; on the Log P/E side, there exists an application programming interface (API) where this data can be pushed into the system.

Since the data moves unidirectionally (from the transactional applications to the C2 applications), the Log P/E API does not allow for the export of data; it only takes data in. Specifically, the exposed methods on the API take in XML data and return back a code indicating if the import was successful.

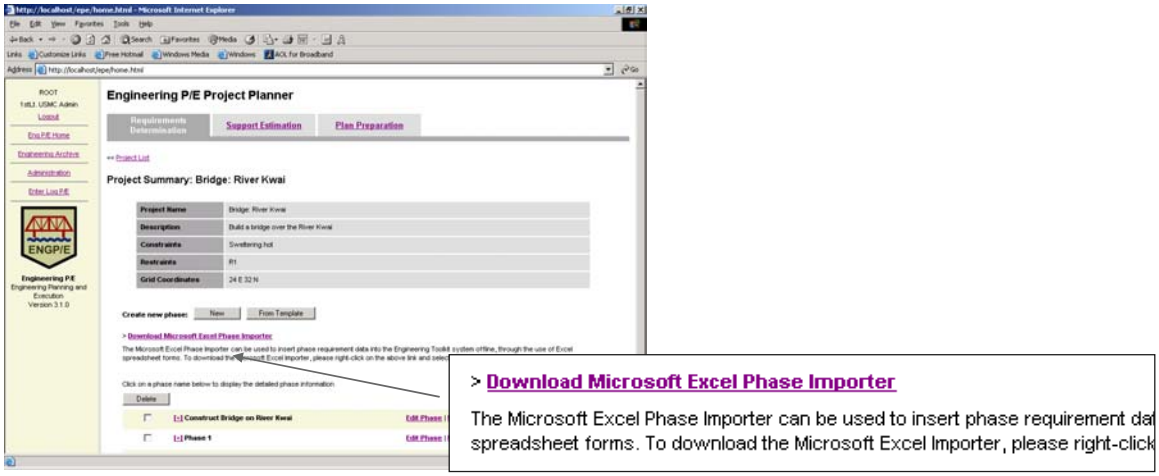
In terms of integration with other systems, this approach is very extensible since it means that Log P/E and Eng P/E can accept C2 data from systems other than ECS and RRTS+. However, a direct call to the Log P/E API also means that the data would not be available to ECS or RRTS+; this may be desirable. However, if the data does need to be available in ECS or RRTS+, then a better approach would be to load the data via the DSDE and move the data from there into Log P/E using this API.

References and Resources

- **Log PE Feeds and APIs Design.pdf** – Detailed technical design document for the Log P/E API.

Bill Of Materials (BOM) Interface for Eng P/E

One of the features of Eng P/E is the ability to load up requirements in the form of a Bill of Materials (BOM) directly into the system via an Excel spreadsheet. Because Eng P/E is a web-based application, this process transfers the data on the spreadsheet (on the client machine) directly into the CLC2S database (which exists on the server). Thus, this mechanism provides another integration point into the system.



Functionally, this process works in a manner very similar to the C2PC Logistics Injector described earlier. The Excel spreadsheet contains an HTTP call to a method on the web server, which accepts the BOM data to be loaded. Thus, the spreadsheet takes the data that has been entered into by the user, converts it to XML and passes that XML to the web server where it is loaded into the system.

As a programmatic interface into CLC2S, this is obviously very specialized, since the XML schema is designed specifically for loading BOM data into Eng P/E. However, this is instructive since it highlights another exposed method for getting data into the system, functionally very similar to the C2PC Logistics Injector.

References and Resources

- **Engineering Toolkit Training Guide.ppt –** User guide for Eng P/E, which explains the usage of the Excel spreadsheet
- **CommonLogisticsCommandAndControlSystem_Submission_Package.zip –** Zip file containing the various XSD files used by the system, including those utilized by the DSDE.

Un-tethered API

The un-tethered functionality is a robust mechanism that allows various disconnected CLC2S systems to interact and share data, even in a degraded communication environment, such as over radios. The un-tethered solution contains application-specific information it uses for messaging, delivery, routing and fail over, but was not designed to interoperate directly with systems apart from CLC2S. (However, once un-tethered data is received by CLC2S, it can certainly be exported using other standard export methods, such as the DSDE).) However, as a potential integration point, it is useful to look at the un-tethered infrastructure, as it does expose a good deal of CLC2S functionality.

Similar to the DSDE, the un-tethered functionality is specifically connected into the transactional applications of CLC2S: ECS and RRTS+. Thus, changes made within those applications can be automatically transmitted as un-tethered messages that will in turn update other disconnected installations of CLC2S. (However, that's not to say that Log P/E is never updated; once those ECS/RRTS+ updates are received by the other CLC2S machine, the Log P/E feed running on that machine will automatically update it's Log P/E database.)

An un-tethered message itself is broken into two parts: the envelope and the body. The envelope information is used in much the same way that it is used in a paper context, in that it contains only the information required to route it to the correct destination. This information is relatively generic and could conceivably be interpreted by systems other than CLC2S.

The body of the message, however, is very specific to CLC2S. It contains XML which precisely corresponds to the format of a CLC2S business object. Thus, that XML would vary depending on the type of message that is being transmitted; a personnel update would be different from a rapid request update, for example. In itself, this is not a problem; however, unlike the DSDE, which uses a fairly generic XML schema, the un-tethered code expects this XML format to be precisely in the format that the business object expects, which is unique to CLC2S. This would be a significant hurdle to overcome if this message needed to be interpreted by another system.

As an integration point with CLC2S, the un-tethered messages are not designed to be portable between systems due to the overhead in interpreting them into a generic format. (Such as what the DSDE does.) However, the un-tethered infrastructure itself could certainly be extended for other purposes; since the routing and forwarding functionality built into the system are designed to only use the envelope information to deliver a message, the contents message body are somewhat irrelevant. So long as the appropriate interface exists on the receiving end to properly interpret that body, the infrastructure could be used to deliver un-tethered messages of several different types.

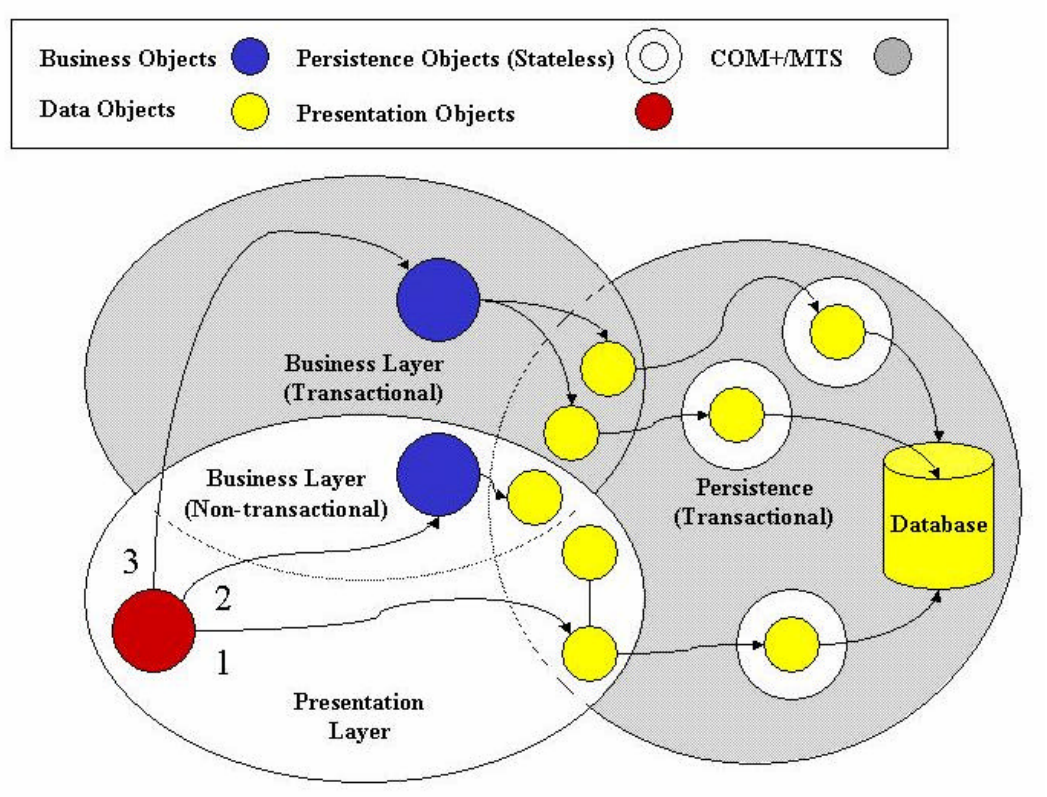
References and Resources

- Untethered SOP.pdf – A Standard Operating Procedure detailing the use of the un-tethered functionality
- Un-tethered Architecture.ppt – User guide and introduction for un-tethered
- Untethered high level technical design.pdf – Technical overview and details

Direct Business Object access

Most of the CLC2S business logic exists in compiled COM objects and DLL files, written in Visual Basic and leveraging Microsoft MTS and IIS to deliver content over a web interface. These utilize the Sapient Microsoft Frameworks, which provide an infrastructure designed to standardize development and offer built-in facilities like logging and persistence management.

The frameworks also help separate business logic from presentation logic, as well as persistence and data objects that can update the database. The two program stacks (ECS/RRTS+ and Log P/E / Eng P/E) each contain a number of Business Objects in the form of compiled COM DLLs which control the various application functions. Because of this, calling the Business Objects (BOs) directly via code represents an additional interface point into CLC2S, albeit a very low-level one. (And potentially dangerous if used incorrectly.)



Interfacing with the CLC2S Business Objects can be very powerful since they represent a gateway to the lowest level of program logic. Also, as compiled DLLs, they can be called from other programs that run on the server machine. However, they also have a number of limitations:

- **Deep technical understanding is required** – Calling a method on DLL is a fairly straightforward process for a system developer; however, it is **critical** to know when and how it is appropriate to call that method since doing it improperly could have serious consequences and destabilize the system. (For instance, calling a method relating to the un-tethered processing when un-tethered has not been configured.)
-
- **Programmatic interfaces exist only on the server –** Any program that calls a CLC2S BO needs to be able to call a COM DLL, which (among other things) means it needs to be either running locally on the CLC2S machine, or operate on a network that allows Distributed COM (DCOM) remote activation calls across its firewalls.
-
- **Regression testing may be difficult –** Because the BO is used for other CLC2S functions, it is critical to know that adding the new functionality will not destabilize existing functionality.

Overall, this approach to interfacing with CLC2S is highly technical and **should not be performed without closely consulting with the CLC2S system development team**. This approach may a powerful way of extending the

system that stops short of new custom development, but it also needs to be very well understood to ensure it does not destabilize existing CLC2S functionality.

References and Resources

CLC2S contains hundreds of Business Objects and a breakdown of all of their related design documentation would be too lengthy to present here. However, each module contains its own documentation with screen details (showing annotated designs of front-end screens), interface specifications and low-level design documents.

A deep overview of the two Sapient Microsoft Frameworks used by CLC2S is located in:

- **Sapient MS Framework Documentation.pdf** – Describes the framework used by the CLC2S C2 applications, Log P/E and Eng P/E
- **Sapient MS Framework v2 Documentation.pdf** - Describes the framework used by the CLC2S transactional applications, ECS and RRTS+

Direct Database Access

Another low-level approach to integrating with CLC2S is to directly access to the Oracle database that is used by the system. Just as accessing the Business Objects gives an interface into the CLC2S business logic, accessing the CLC2S database directly can give a powerful view into the lowest workings of the system. However—just as with the BOs—this can also seriously destabilize the system if not done carefully.

As noted earlier (and displayed in Figure 3), the CLC2S suite of applications can be broken down into three main architectural areas:

- Transactional Applications (ECS and RRTS+)
- Command and Control Applications (Log P/E and Eng P/E)
- Communications Infrastructure (Un-tethered and DSDE functionality)

Although CLC2S uses a single Oracle database to hold all its data, it is logically subdivided into three sections that correspond to these areas (ECS, SUL and COMM) to keep these modules functionally discrete. Therefore, if an application is able to access the Oracle database (via ODBC, for example), it can take advantage of a raw, undiluted look into the CLC2S data.

As noted above, modifying this data directly could have serious impacts on system stability and performance and **should not be performed without closely consulting with the CLC2S system development team**. If possible, the safest approach to viewing this data is to use the DSDE or reporting database, as noted earlier. Failing that, the best course of action would be to limit database access to a read-only view of the tables and their data.

References and Resources

The supporting documentation for the three sections of the Oracle database are all very similar. Each contains an Entity/Relationship diagram (in ERWin format) that describing the table relationships as well a written data dictionary that details the meanings of the various tables and columns:

E/R diagrams:

- **CIS_SCHEMA.ER1**
- **COMM.ER1**
- **SUL2_Schema.ER1**

Data Dictionaries:

- **USMC_Data_ Dictionary_COMM.pdf**
- **USMC_Data_ Dictionary_ECS.pdf**
- **USMC_Data_ Dictionary_SUL.pdf**

Integration Approaches

Because “integration” with CLC2S can take many forms, it is critical to consider the goals of such an integration, as well as the type of system that it is being integrated with. At a high level, integration may take one of two approaches:

- Data integration
- Programmatic integration

Data integration simply involves the sharing of data between CLC2S and an external system. At this level, both systems can likely continue to operate as independent entities, sending data back and forth as necessary, usually asynchronously. The Deployed SDE and Reporting components of CLC2S are specifically designed to support this type of integration, and can be used with no modifications to the system.

Programmatic integration assumes a more synchronous interface with the external system is required. This would likely involve some specialized coding of the external system, since it would need to call one of the various CLC2S APIs to make the call and/or retrieve the relevant data. Integration of this type might be appropriate when developing functionality to extend CLC2S, or to integrate it closely with another system. (Such as was done with the C2PC injector.)

Of the two approaches, data integration is by far the safest and most robust, since there are existing tools and processes existing within CLC2S that are designed to support this. Programmatic integration provides more flexibility in terms of the level of integration, but it also requires a great deal more overhead in development and testing, and therefore invites more opportunities for destabilizing the existing system.

Future Steps

As CLC2S evolves, integration with other Navy and Marine Corps systems will continue to be a priority. Furthermore, the addition of new CLC2S functionality may require new development that integration with the existing modules of the system. As these new functions and systems are identified, they will likely require modifications to the existing interfaces, or even suggest additional interfaces.

Such additional interfaces may include:

- **Enhanced support for synchronous data interfaces** - for instance, a web service component that allows other systems to call the CLC2S server machine via an HTTP interface.
- **Improved export functionality for Log P/E and Eng P/E data** – currently much of the data-level integration is focused on getting transactional ECS data that is fed into the C2 applications; it may be useful to extend the ability to directly import or export C2 information directly from CLC2S, such as mission planner data.
- **Creation of an "interface toolkit"** – just as the DSDE provides a user-friendly, configurable front end for data interfaces to CLC2S, create a similar set of functions for the programmatic interfaces, abstracting the Business Objects into more generic formats.